VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Testování systému na platformě Android a PHP5

# System testing on the Android and PHP5 Platform

Ostrava 2013 Tomáš Holák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Tomáš Holák**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Testování systému na platformě Android a PHP5
System Testing on the Android and PHP5 Platform

Zásady pro vypracování:

Cílem závěrečné práce je vytvořit testy na základě rozličných testovacích praktik v jednotlivých úrovních testování a jejich posouzení z hlediska pracnosti, rozsahu pokrytí, úspěšnosti, vhodnosti apod. pro komunitní systém firmy SCOVECO, s.r.o. V současnosti existuje klientská část, která je na platformě Android, a serverová část, která je na platformě PHP5.
Postup zadání:
1.Vytvořte popis jednotlivých testovacích technik pro různé úrovně testování.
2.Vytvořte testovací případy a proveďte testování
3.Zhodnoťte dosažené výsledky z hlediska pracnosti, rozsahu pokrytí, úspěšnosti, vhodnosti, jejich vzájemné posouzení.

Seznam doporučené odborné literatury:

[1] GALIN, D. Software Quality Assurance: From Theory to Implementation. 1 ed.: Addison-Wesley, 2003. 616 p. ISBN 0201709457.
[2] MCCAFFREY, J.D. Software Testing: Fundamental Principles and Essential Knowledge. BookSurge Publishing, 2009. 118 p. ISBN 1439229074.
[3] PATTON, R. Software Testing. 2 ed.: Sams Publishing, 2005. 408 p. ISBN 0672327988.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Kožusznik, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

*Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.*

V Ostrave, 7.5.2013

Tomáš Holák

*Súhlasím so zverejnením tejto bakalárskej práce podľa požiadaviek čl. 26, odst. 9 Študijného a skúšobného poriadku pre štúdium v bakalárskych programoch VŠB-TU Ostrava.*

V Ostravě, 7. 2. 2013

Jan Kožusznik

Kožusznik Jan

Rád by som poďakoval pánovi Ing. Janovi Kožusznikovi, PhD., za odborné vedenie, cenné pripomienky a rady využité pri spracovaní tejto bakalárskej práce a taktiež za ochotu o daných problémoch diskutovať.

**Abstrakt**

Táto bakalárska práca popisuje vstup automatizovaného testovanie do procesu vývoja softwarového systému. Venuje sa aktuálnym technológiám ako je Facebook a mobilné aplikácie pre Android.

Prvá časť práce popise štruktúru systému, ktorý sa bude testovať. Ďalej vykresľuje problematiku nastavenia testovacieho prostredia, ktoré môže zabrať veľkú časť času prideleného na proces testovania. Druhá časť sa venuje samotnej tvorbe automatizovaných testov pomocou zvolených nástrojov. Architektúra testovaného systému je klient - server a aj samotné testy sú tvorené vo dvoch úrovniach.

Záverečná časť hodnotí dosiahnuté výsledky, špecifikuje nedostatky a navrhuje zlepšenia a budúcu prácu.

**Kľúčové slová**

Android, Automatizácia testov, Emulátor, Integračné testovanie, PHP Server, PHPUnit, Regresne testovanie, Robotium, Testovanie systému.

**Abstract**

This thesis describes the entrance of test automation into a process of software development. Work is related to the latest technologies like the Facebook and Android powered mobile applications.

The first part describes the structure of the system under tests. It shows the efforts needed to configure the test environment. The second part solves the implementation of automated tests using selected tools. Architecture of application under test is client - server and there are tests built for both levels.

Conclusion of this work evaluates the achievements, shows the shortcoming and proposes improvements and the future work.

**Key Words**

Android, Emulator, Integration testing, PHP Server, PHPUnit, Regression testing, Robotium, System testing, Test automation.

# List of symbols and abbreviations

| | |
|---|---|
| apk | Android Package file |
| app | Application |
| ADB | Android Debug Bridge |
| ARM | Advanced RISC Machine |
| ASTQB | American Software Testing Qualifications Board |
| ATC | Automated Test Case |
| AUT | Application Under Test |
| CI | Continuous Integration |
| CPU | Central Processing Unit |
| CPU0 | The first CPU's core |
| CRUD | Create, Read, Update, Delete |
| DOM | Domain Object Model |
| FB | Facebook |
| GUI | Graphical User Interface |
| HAXM | Hardware Accelerated Execution Manager |
| HTTP | HyperText Transfer Protocol |
| ID | Identification key |
| IDE | Integrated Development Environment |
| IP | Internet Protocol |
| ISTQB | International Software Testing Qualifications Board |
| ISO | Compact Disc File System |
| MM | MeetMe |
| OS | Operating System |
| PHP | PHP: Hypertext Preprocessor |

| | |
|---|---|
| RISC | Reduced Instruction Set Computing |
| QA | Quality Assurance |
| SDK | Software Development Kit |
| TC | Test Case |
| TS | Test Suite |
| UC | Use Case |
| VM | Virtual machine |
| XML | Extensible Markup Language |
| XML-RPC | XML- Remote Procedure Call |

# Content

# 1 INTRODUCTION

Nowadays I've been working as Quality Assurance Engineer for almost three years. I'm also quite interested in this area therefore I decided to negotiate a bachelor thesis related to this area.

An opportunity came to me with an offer from Ing. Jan Kožusznik, Ph.D., a Chief technical officer of the Scoveco company. A subject of my work is the MM software system providing possibility to schedule informal meetings via Android mobile application.

My task here is to create a testing framework for this application, focusing on possibility of rerunning implemented tests automatically. The tests covering the basic functionality should be implemented for both parts of this application, i.e. client and server. This tests will be run after each bigger change within this application and they should guarantee that the basic features are still working fine and there has been no unwanted changes interfering involuntary parts of application introduced. Results of analysis of application's code, analysis of running application and a brief specification provided will be a basic domain knowledge for tests creation.

Not less challenging part of this work is to setup a testing environment, covering a configuration to get working application itself and also setup of testing tools. From my practice of Quality Assurance Engineer I've got experiences with configuration issues, amount of time and effort it can cost.

Conclusion part of this work contains evaluation of used tools and suitability of test automation per each component of this application where it has been implemented. There is also a research how to improve the testing process and an attempt to apply testing metrics to get a measurable results for the level of test automation.

# 2 ABOUT TESTING

I've already been certified by ISTQB Foundation Level and I've seen a nice introduction to testing in their syllabi:

*"Software systems are integral part of life, from business applications to consumer products. Most people have had an experience with software that did not work as expected. This can lead to many problems, including loss of money, time or business reputation and could even cause injury or death.*

*A human being can make a mistake, which produces a defect in the program code or in a document. If a defect in code is executed, the system may fail to do what it should do (or do something it shouldn't), causing a failure. Defects in software, systems or documents may result in failures, but not all defects do so.*

*With the help of testing, defects could be found and subsequently fixed before software released for real live usability and it is possible to measure the quality of software in terms of defects found, for both functional and nonfunctional software requirements and characteristics."* [1; p. 11]

## 2.1 TEST PROCESS AND BASIC PRINCIPLES

This process (Figure 1) is carried out throughout the whole lifecycle of software development. There are several basic principles that should be remembered for it. The ASTQB has picked out this:

**"Seven testing principles:**

- ***Principle 1 - Testing shows presence of defects***
  *Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.*
- ***Principle 2 - Exhaustive testing is impossible***
  *Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts.*
- ***Principle 3 - Early testing***
  *To find defects early, testing activities shall be started as early as possible in the software or system development life cycle, and shall be focused on defined objectives.*

- ***Principle 4 - Defect clustering***

  *Testing effort shall be focused proportionally to the expected and later observed defect density of modules. A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures.*

- ***Principle 5 - Pesticide paradox***

  *If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects. To overcome this "pesticide paradox", test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to find potentially more defects.*

- ***Principle 6 - Testing is context dependent***

  *Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site.*

- ***Principle 7 - Absence-of-errors fallacy***

  *Finding and fixing defects does not help if the system built is unusable and does not fulfill the users' needs and expectations* ." [1; p. 14]

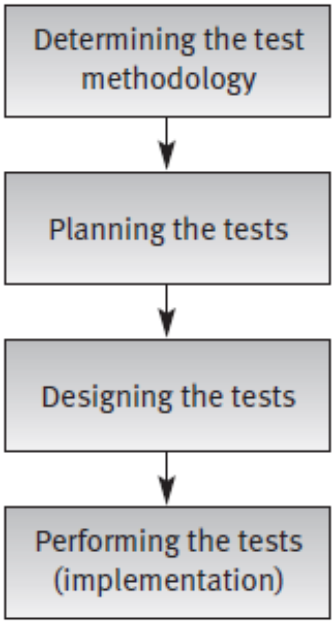



**FIGURE 1: TEST PROCESS [2; P. 218]**

## 2.2 COMMON TESTING TECHNIQUES OVERVIEW

There are two main testing classes based on two different concepts of testing software:

- **Black box** - testing that does not care about the internal calculation of a system. The tester usually does not have access to system's source code.
- **White box** - tester has access to the source code, testing cares about internal mechanism of a system.

This two main classes are extended more to provide various testing types that are applied during a testing process:

- **Unit testing** - White box
  *"Unit testing is the testing of individual hardware or software units or groups of related units."* [3; p. 39]
- **Integration testing** - Black and/or White box
  *"Integration test is testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them."* [3; p. 39]
- **Functional and System testing** - Black box
  *"Functional testing involves ensuring that the functionality specified in the requirement specification works. System testing involves putting the new program in many different environments to ensure the program works in typical customer environments with various versions and types of operating systems and/or applications. System testing is testing conducted on a complete, integrated system to evaluate the system compliance with its specified requirements."* [3; p. 39]
  There are also non functional requirements on software system and since it is already done in this phase also this testing classes should be executed.
  - ***"Stress testing*** *– testing conducted to evaluate a system or component at or beyond the limits of its specification or requirement."* [3; p. 39]
  - ***"Performance testing*** *– testing conducted to evaluate the compliance of a system or component with specified performance requirements."* [3; p. 40]
  - ***"Usability testing*** *– testing conducted to evaluate the extent to which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component."* [3; p. 40]

- **Acceptance testing** - Black box

  *"Formal testing conducted to determine whether or not a system satisfies its acceptance criteria (the criteria the system must satisfy to be accepted by a customer) and to enable the customer to determine whether or not to accept the system."* [3; p. 40]
- **Regression testing -** Black and/or White box

  *"Regression testing is selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements."* [3; p. 40]
- **Beta testing -** Black box

  *"When an advanced partial or full version of a software package is available, the development organization can offer it free to one or more (and sometimes thousands) potential users or beta testers. These users install the software and use it as they wish, with the understanding that they will report any errors revealed during usage back to the development organization."* [3; p. 41]

All this basic knowledge about testing is the starting point for our testing project. The next chapter describes the software system under tests and according its specification and regarding the requirements from developers site the appropriate testing tools and techniques will be selected.

Note: Static testing techniques are out of scope of this work.

# 3 APPLICATION UNDER TEST

The MM system consists of two parts, i.e. client and server. This is a very common model of software systems where the client makes a service requests from another program which fulfills this request - a server (Figure 2).


FIGURE 2: ARCHITECTURE SIMILAR TO THE MM SYSTEM [4]

## 3.1 SERVER PART

A program written in PHP, a server-side scripting language originally created by Rasmuf Lerdof, the current version is being produced by The PHP Group. This program needs to be deployed into a web folder of a PHP server (the Apache web server combined with PHP) and can be executed by another programs (clients) through the network [5].

Communication between server and client is provided by XML-RPC, i.e. Remote Procedure Calling protocol working over the Internet. The client asks the server to execute a procedure and the server returns the response in xml format. Client's message is a HTTP-POST request [6].

Data persistency is provided by MySQL, i.e. an open source relational database management system, which completes the LAMP software stack. The LAMP provides a platform for development and deployment for web based applications [7].

## 3.2 CLIENT PART

A mobile application running on Android OS. Primarily controlled by a touch screen, written in Java. Android SDK tools compile the code and the other source data into an Android package, an archive file with .apk suffix. Android powered device uses this single file to install the application [8].

Android OS is a Linux based operating system, currently with the biggest worldwide smart phone OS market share [9]. It's been developed within The Android Open Source Project that provides the full open source software stack for developing Android applications [10].

## 3.3 FACEBOOK - THE SOURCE OF USER ACCOUNTS FOR MM SYSTEM

A software system providing interaction between people needs to register user accounts. Architecture of this application does not allow independent registration but a potential user needs to have an existing registration on FB, nowadays the most popular social network that had surpassed a billion active users in 2012 [11].

### 3.3.1 FB APPLICATION

To gain the FB accounts, the application needs to be integrated with FB platform. A way how to do it is the FB SDK. It supports reading and writing to the FB API and also supports for user's login with FB authentication [12].

Two main parts of this integration are:

- Installation of the FB SDK and its import into an IDE, usually Eclipse is a popular choice for Android applications.
- Creation of FB app on the App Dashboard located on the FB Developers site, a developer assigns an app name and he/she also gets an app ID for his/her app there [12].

For my tests I've registered a FB App using my private FB account on the App Dashboard (Figure 3). Important info is the "App ID" and the "App Secret" that is used in the process of App's authentication when attempting to cooperate with the FB.

The FB App for tests (Figure 3):

- App ID: 386072568157506
- App Secret: 8a10951a4ac732adfd5985e92cf448c9

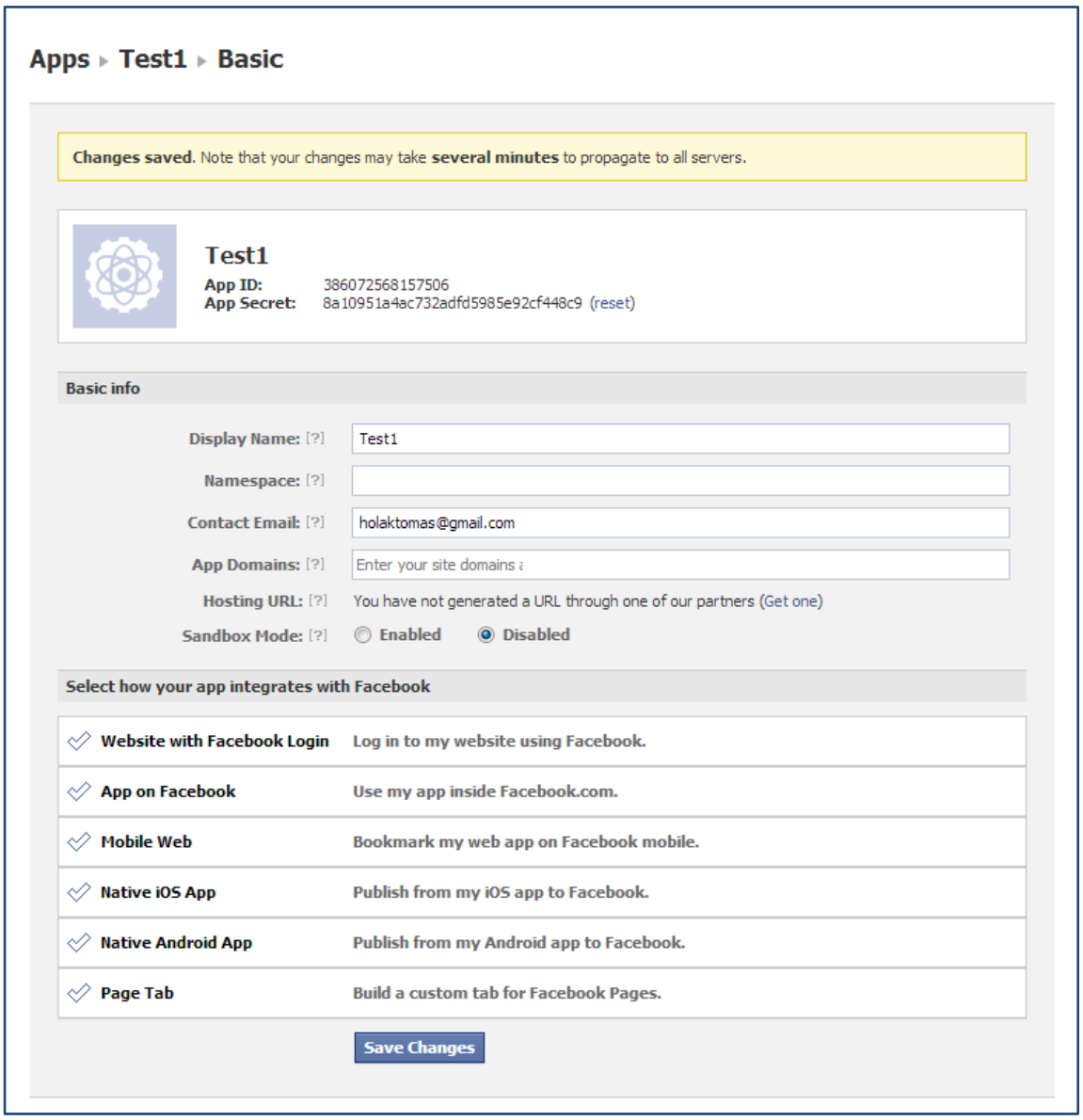



FIGURE 3: THE FORM FOR FB APP REGISTRATION

### 3.3.2 FB LOGIN

Facebook offers several login flows for different devices and projects. Each of these flows use the OAuth 2.0 standard, an open protocol providing a secure authorization in a standard method from desktop, mobile and web application [13].

Our application uses the Client-side JavaScript SDK to integrate a login with a cooperation of a JavaScript-capable browser. The JavaScript SDK manages every access token generated during the login process, the tokens are persisted and Graph API calls are signed automatically [12].

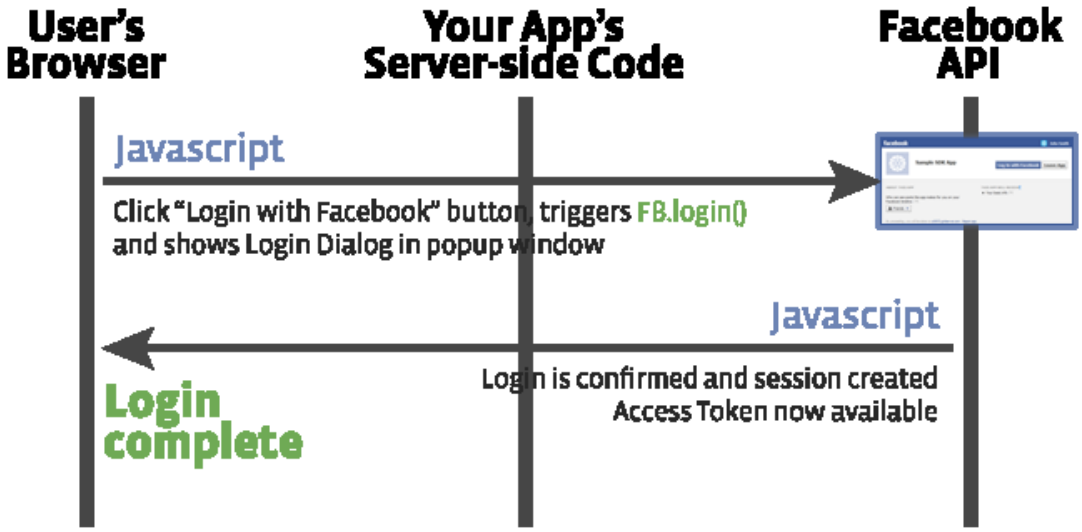



FIGURE 4: FB LOGIN ARCHITECTURE USING JAVASCRIPT [13]

### 3.3.3 TEST DATA - FB ACCOUNTS

Test for MM system should interact with FB system. There is a need for valid FB accounts for this purpose. When a user connects with an app using FB login, the app needs to obtain an access token to be able to work with FB APIs.

**Access Tokens**

A random string identifying a User, App, or Page session. It contains info about [13]:

- granted permissions
- token's expiration
- the app which generated this token

There are different types of access tokens to support various cases. For the purpose of MM system, the User Access Token is needed for app calls an API to read, modify or write a specific person's FB data. "*In essence it is a temporary password that the app can use on behalf of the person.*" [13]

For the client's part testing the FB login form is included in the test's execution flow therefore the access tokens are handled automatically via JavaScript SDK. But for the server part testing the access tokens needs to be provided - in the code of automated tests . The Figure 5 shows the FB developers page where to generate user access token [13].

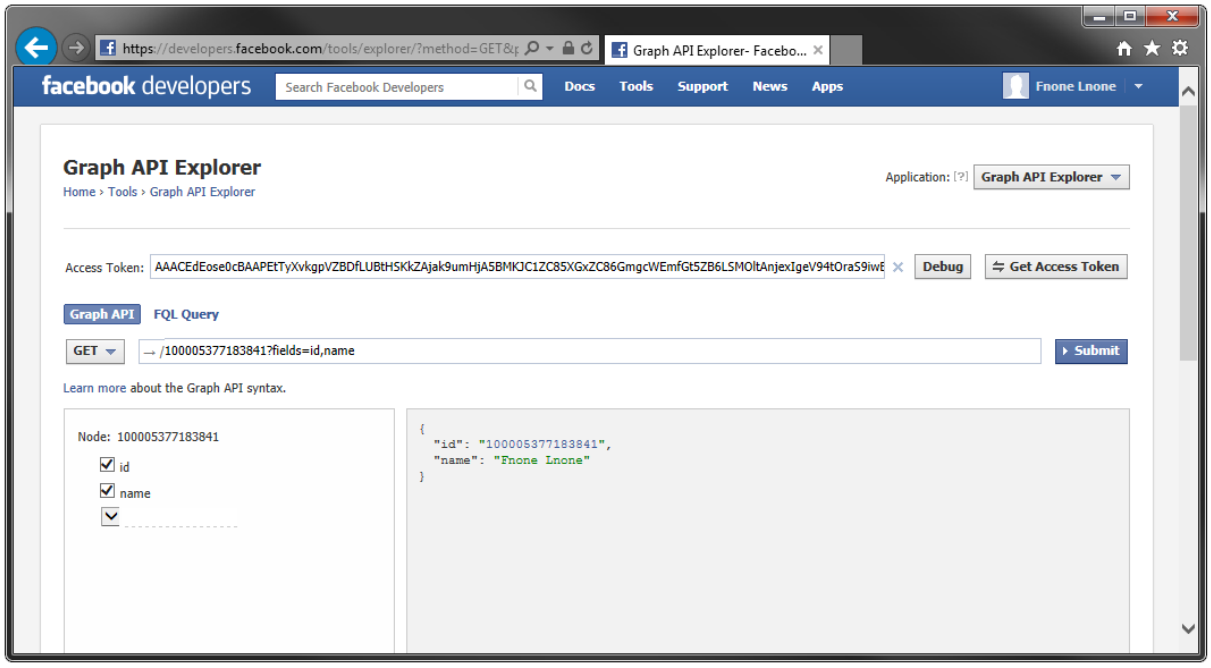



FIGURE 5: GENERATING THE USER ACCESS TOKEN

### Extending validity period

Access tokens have usually a validity period of about one or two hours. They need to be extended in order to continue using these tokens. If the login flow is triggered then this token refreshes and extends its validity. This needs to be done manually for server side tests.

HTTP request to receive extended token [13]:

*https://graph.facebook.com/oauth/access_token?grant_type=fb_exchange_token&client_id=APP_ID&client_secret=APP_SECRET&fb_exchange_token=SHORT_LIVED_ACCESS_TOKEN*

**The test accounts**

- E-mails:
  - hol1@centrum.sk
    - password: butter
  - hol4@centrum.sk
    - password: butter
- FB accounts:
  - hol1@centrum.sk
    - password: butter
    - access token:
      - AAFfIYZAIVUIBAP0Y1HcqYxPegdwm9TUvcZCP1qDQWNs66pbbMAYmRLbkqiGuVWsdlgEvSytmU77FZBk6nYnBtIkqF3lcRJZBpBFxJC99AZDZD
  - hol4@centrum.sk/butter
    - password: butter
    - access token:
      - AAAFfIYZAIVUIBALANay4SHMtyEJYyyY1sokUWI71GR4gY59blV68sypFLG52ykreLTn2T1CVZBwPzbUBUZAQGt5P8KsJxvtCeD1pBUzoAZDZD

# 4 TEST PLAN AND STRATEGY

In the process of software development it is very important to catch defects early since this are cheaper to correct. Defects are also introduced while another defects are being fixed. This are called regression bugs and regression testing's intent is to ensure that changes and fixes have not introduced any of them.

One of the main reasons for regression testing is to determine whether a change in one part of the software affects the other parts. Common methods of regression testing include rerunning previously completed tests and checking whether program behavior has changed [3].

Since the rerunning process for regression testing, the automation of test scenarios can be a good practice how to reduce costs and also make testing in this area more precise. Costs reduction is related to human hours spent on this activity and higher precision(compared to tests executed by a human) is related to work of machine which cannot go wrong when comparing two strings of characters, for example.

This is also the original idea for tests creation for the MM system, i.e. to create a set of automated regression tests. Section 4.2 describes the planning process which is only a fraction comparing to the planning for test process running through the whole development cycle. *"The test plan describes the scope of the overall test effort and provides a record of the test planning process. It identifies the features and test environments to be tested, the entry and exit criteria to be used, quality goals, and other items."* [14]

## 4.1 INPUTS FOR THE TEST PLAN AND THE TESTING ITSELF

To get started with the tests there is a need to have the subject of testing available - the software system and also the expected behavior of this system must be known and documented.

### 4.1.1 AUT SPECIFICATION

The first input for my tests. It contains a text that briefly describe this software system:

- Functional requirements, a definition what the system should to do but not defining how it should to do, i.e. it is solution independent.
- List of UCs and their description. An UC is a set of steps, defining interactions between an actor and a system, to get a wanted results.
- Releases description, a brief description of already released versions.

- Client's GUI description, a list of screens which a user can see while using this application, there is also a description per each screen with possible inputs and outputs.

This document will be used to derive the TCs and they will be automated later.

### 4.1.2 AUT SOURCE CODE

Application's source code will be used for enhanced analysis of internal system logic and also to get working system - ready for automated tests development and execution. Storage, management and sharing of this code between the development team is controlled by Git - free and open source distributed version control system [15].

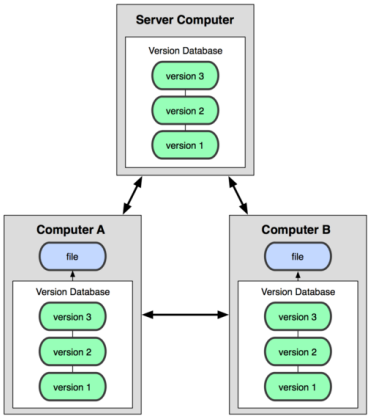



FIGURE 6:GIT ARCHITECTURE [15]

The basic steps how to get the source code into my workstation to be able to work:

- Install a Git client.
- Generate own ssh public key using ssh-keygen since the ssh authentication method is used to establish a secure connection to Git server.
- Provide this ssh public key to the Administrator of Git server, he will add You as an user of git and he will also append the key to the *authorized_keys* folder.
- Clone the remote repository.
  - *git clone ssh://gitolite@cml1.scoveco.cz:/mm*
- Check out a wanted branch.
  - *git checkout iss243*
- Commit updates on code when needed.
  - *git commit -a -m "message"*
- Publish your commits by pushing your branch to the remote location
  - *git push origin iss243*

## 4.2 TEST PLAN SPECIFICATION

The following are the main points of our high level test plan:

- The subject of testing (AUT) is MM software system, specifically the selected classes of the server part and the GUI of the client.
- The goal is to create automated test scripts checking the basic functionality of the MM system.
- The first activity is to configure the environment of the MM system.
- There is only one responsible resource - me, with possibility to consult things with a developer of the MM system.
- Test strategy for the MM server - Integration testing of selected classes, selected tool - PHPUnit.
- Test strategy for the MM client - System testing of the whole MM system through client's GUI, an attempt to automate the manual TCs.

### 4.2.1 TC SPECIFICATION

There is a standard (IEEE 829) describing what everything should the TC specification contains. It contains many details and following this suggested level could lead to writing at least a page long descriptive text for each TC. If possible, taking the shortcuts is a good way for better efficiency [30].

The following text represents the high level TC specification for the MM system:

- Common rules
    - Actor is an user of MM system with a valid FB account.
    - Environment is Android powered device with Internet connection.
    - The obvious expected results are not written.
- TC1.1: Login.
    - Covering UC1.3, UC1.4, UC1.5
    - Flow:
        - Actor starts the MM app.
        - Actor clicks the FB button → FB login form appears.
        - Actor fills in valid FB credentials and clicks the Login button.
        - Expected: My Events view is shown.

- TC2.1: Create a new Event.
  - Covering UC1.1
  - Flow:
    - Actor logs in (TC1.1).
    - Actor clicks the Create Event button → Event details view appears.
    - Actor specifies the Event's details.
    - Actor click the Save Event button.
    - Expected: New Event has been created, it is shown in the list of My Events.
- TC2.2: Update an Event.
  - Covering UC1.1
  - Flow:
    - Actor creates an Event (TC2.1).
    - Actor opens the already created Event.
    - Actor updates any Event details.
    - Actor click the Save Event button.
    - Expected: Event has been updated, reopen the Event's details and check updated values.
- TC2.3: Delete an Event.
  - Covering UC1.1
  - Flow:
    - Actor creates an Event (TC2.1).
    - Actor clicks and holds on the already created Event's item in My Events → Dialog to delete the Event appears.
    - Actor confirms the dialog.
    - Expected: Event is not present in the list of My Events.
- TC3.1: Actor accepts the invitation.
  - Covering UC1.2
  - Flow:
    - Actor1 creates an Event (TC2.1).
    - Actor1 sets the Actor2 as invited for this Event (TC2.2).
    - Actor1 logs out.
    - Actor2 logs in to the MM app.

        - Actor2 navigates to the list of Friends Events (scroll to the right from My Events View) → the Event is shown in the list.
        - Actor2 accepts this invitation by clicking the tick button shown on the Event's item.
        - Expected1: Accepted Event is still shown in the list of Friends Events.
        - Expected2: Actor1 can see the number of accepted invitations for this Event has increased.
- TC3.2: Actor declines the invitation.
    - Covering UC1.2
    - Flow:
        - The same flow as for TC3.1 but the last step where Actor2 accepts the invitation.
        - Actor2 declines this invitation clicking the cross button shown on the Event's item.
        - Expected1: Declined Event disappears from the list of Friends Events.
        - Expected2: Actor1 can see the number of declined invitations for this Event has increased.

## 4.3 ABOUT SELECTED TESTING TECHNIQUES

### 4.3.1 TEST AUTOMATION

*"Automated delivers software tests provide three key benefits: cumulative coverage to detect errors and reduce the cost of failure, repeatability to save time and reduce the cost to market, and leverage to improve resource productivity."* [16; p. 5] These are benefits for automation for comprehensive testing. In our case we need to create a set of basic test assuring that the basic system functionality has not been corrupted by any updates during following development process.

Someone can see the test automation as record and playback the tests but this does not result in a robust and maintainable test library. The other extreme is programming of complex scripts that anticipate program's behavior and provide a response for all possible situation. Tests complexity increases risk of bugs in tests and we need to avoid the situation the tests are objective for testing too.

### 4.3.2 INTEGRATION TESTING

*"While unit tests deal with small units of software or modules, integration tests deal with several units that combine into a subsystem. System tests refer to the entire software package/system."* [3; p.220] For unit testing there are several tools available and the same tools can be also used for integration tests creation:

- SimpleTest
- PHPUnit

Both of them belong to the family called xUnit. "*The origins of these frameworks actually started in Smalltalk. Kent Beck build a simple framework to organize and run unit tests. The focus was on making it easy for programmers to define the tests using their regular smalltalk environment, and then to run either a subset or a full set of tests quickly."* [17]

After a small research on the Internet and from recommendation of the Supervisor of my thesis I've decided to use PHPUnit.

### 4.3.3 SYSTEM TESTING AUTOMATION

A typical practice for System testing is performing TCs manually. A tester walks through the TCs, executes the test steps and checks expected results.

Nowadays there are many frameworks available that can substitute tester's work and "click over the application" automatically.

From my IT profession I'm experienced with System testing for web applications. Tools for this area are really mature in this days and it is not hard to write reliable and robust test scripts. The Selenium is well known tool in this area.

There are different tools used when it comes to native applications testing. It is possible to record manual tests using *SeeTest* (http://experitest.com/) or *eggPlant* (http://www.testplant.com/). Afterwards, the recorded scripts can be exported to almost any programming language, updated and run again.

On the other side, there is Robotium. It does not allow the user to record manual test scenarios but it provides an API that allows the user to interact with the app under tests. After a small research on the Internet I've decided to use the Robotium.

# 5 TEST IMPLEMENTATION FOR THE SERVER

The task here was to create tests scripts for two chosen classes:

- com.*scoveco.mm.php/application/dao/doctrine/SCDaoDoctrine.php*
- *com.scoveco.mm.php/application/sccore/SCAppCore.php*

The aim of testing are the be public methods located in mentioned classes since they provide the API of this classes and actually this needs to be tested.

## 5.1 SETUP TESTING ENVIRONMENT

To create tests here I need to setup working server part of this software system. Two main things need to be done here:

- Web server installation with all needed add-ons.
- The application's server scripts deployment into an already installed and running web server's web folder.

### 5.1.1 WEB SERVER CONFIGURATION

The server part will be created as a virtual machine located on my workstation. A few steps are needing to install a LAMP  and additional tools to get working the whole MM system. The following overview of actions also covers configuration to enable debugging:

- Install host for a virtual machine (VM) → VMware player 5.0.1
- Install web server environment →LAMP, a software stack that consists of:
  - Operating system → Ubuntu 11.04 server
  - Http server → Apache2
  - Data storage → MySQL
  - Server site scripting language and parser → PHP5
- Web server should work fine now, this can be checked in a web browser exploring IP address of already created VM.
- Install openssh server to be able to connect to the server using Putty and WinSCP
- Add write permissions to the Ubuntu user for the web folder (*/var/www/*) to be able to simply deploy MM server scripts to the web server.
- Install Doctrine, a persistence service and other related tools provider.

- Setup permissions for Doctrine proxy folder.
- Install cUrl, a command line tool for transferring data with URL syntax, used for communication between MM server and FB [20].
- Install PHPUnit to be able to run PHPUnit test scripts on the web server.
- Install Zend debugger to be able to debug the test scripts.
- Install XDebug.
- Install MySQL and PHPMyAdmin.
- Install XML-RPC to enable communication between the MM client and the MM server:
    - *sudo apt-get install php5-xmlrpc*
    - enable extension in php.ini
- Setup permissions for the web server log folder for users www-data and PHPUnit
- Restart Apache2:
    - *sudo service apache2 restart*

### 5.1.2 IDE FOR TEST SCRIPTS DEVELOPMENT

**Basic configuration**

- Install IDE → Eclipse3.8.
- Import projects from the local git repository into the IDE.
- Create a new source folder for tests in the MM server project.
- Solution for synchronization of the MM server scripts placed in workspace and the web server's web folder is WinSCP with a its feature *Keep remote directory up to date*.

**Debugging php scripts**

IDE and web server connection is solved by an SSH tunnel between the IDE's host machine and the web server using Putty [21]:

- Host IP and port → 192.168.179.129:22
- Connection type → SSH
- SSH Tunnel
    - L8010 → 127.0.0.1:80
    - R10008 → 127.0.0.1:10008

**Eclipse running/debugging configuration**

Server part (Figure 7):

- PHP server → http://localhost:8010
- File (a script to be debugged) → Path to file in local repository.
- URL (a path to script on the server) → Auto Generate (Eclipse can automatically generate this correctly if the whole project folder placed into web folder on server).

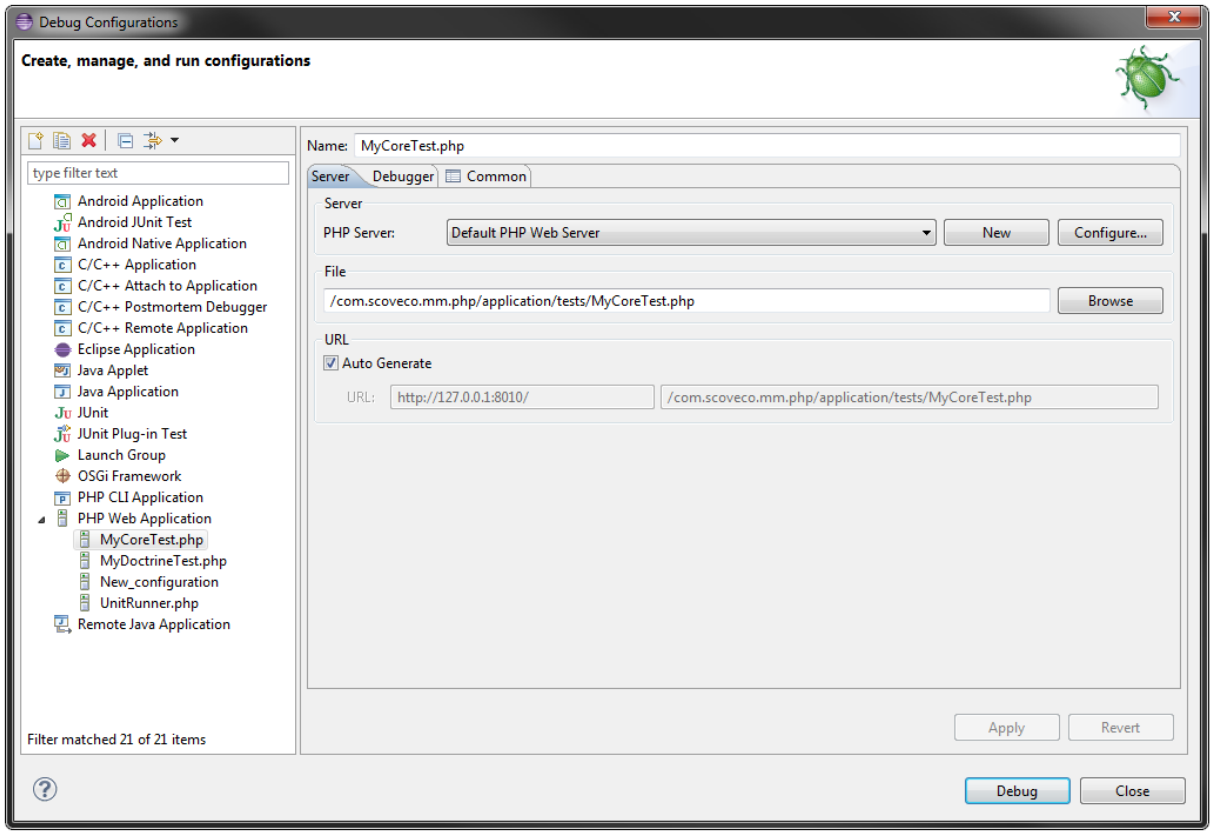



FIGURE 7: DEBUG CONFIGURATION → SERVER

Debugger part (Figure 8):

- Server Debugger → Zend Debugger
- Port → 10008
- Client Host/IP → 127.0.0.1
- Debug Response Timeout → default value (50000ms)

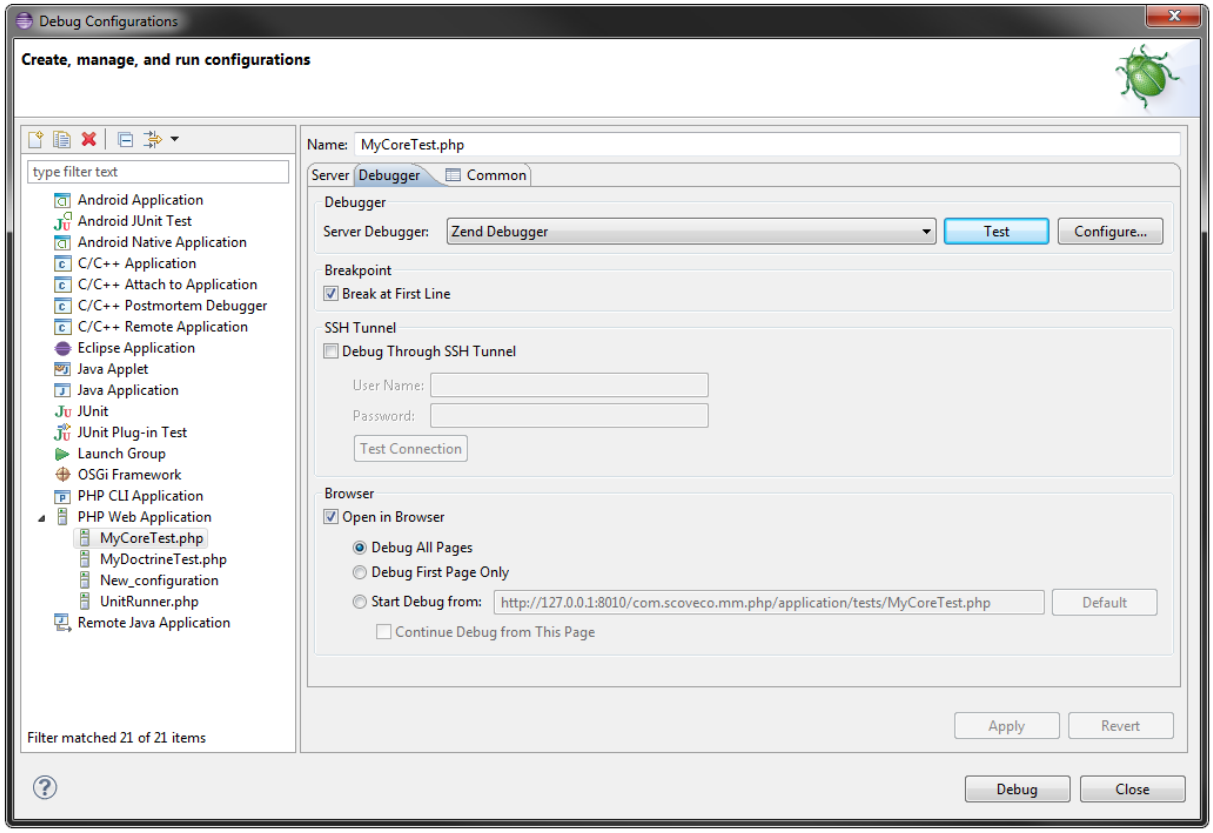



FIGURE 8: DEBUG CONFIGURATION → DEBUGGER

## 5.2 IMPLEMENTATION

PHPUnit is primarily used for unit testing of classes on the basic level. Although our chosen classes to tests mainly integrates nontrivial logic from another classes, then PHPUnit can be still used here. There are few principles that should be followed when writing test [22]:

- Tests should be easy to read and write.
- Easy to execute.
- Isolation, i.e. the tests should not affect each other.
- Testing based on publicly visible behavior.
- Fine-grained tests testing one aspect of one object.
- Descriptive test names that can be used like a documentation.

### 5.2.1 TEST PROJECT SETUP AND TEST SCRIPTS OUTLINE

There won't be a new project for tests, the tests will be placed into a folder within the MM server project, for simplified access to mandatory imports, as described in Section 5.1.2.

The tests are PHP scripts that follow the basic conventions [22]:

- The tests for a class Class go into a class ClassTest.
- ClassTest inherits from PHPUnit_Framework_TestCase.
- The tests are public methods that are named *test**.
- Assertion methods are used to assert that an actual value matches an expected value. These are placed inside the test methods.

Example:

```
<?php
require_once 'PHPUnit/Framework.php';

class ArrayTest extends PHPUnit_Framework_TestCase
{
    public function testNewArrayIsEmpty()
    {
        // Create the Array fixture.
        $fixture = array();

        // Assert that the size of the Array fixture is 0.
        $this->assertEquals(0, sizeof($fixture));
    }

    public function testArrayContainsAnElement()
    {
        // Create the Array fixture.
        $fixture = array();

        // Add an element to the Array fixture.
        $fixture[] = 'Element';

        // Assert that the size of the Array fixture is 1.
        $this->assertEquals(1, sizeof($fixture));
    }
}
?>
```

Additionally, all the needed scripts needs to be imported. Also a fixture needs to be set before the test execution. It is a known an expected state of all components that can affect the results of test (data in DB, etc.). Methods *setUp()* and *tearDown()* are used for this purpose [22].

### 5.2.2 CODE COVERAGE

Testing metrics help us to find out what is being tested. PHPUnit also provides a functionality that provides an insight into what parts of the code are executed during tests run.

**Configuration**

- XDebug needs to be installed but it overrides the Zend Debugger if loaded as *zend_extension*, it needs to be loaded as *extension* in php.ini [23].
- Filter for code coverage needs to be set in phpunit.xml otherwise all executed files are counted in report [22].
- Logging path in phpunit.xml, the html files for reports are generated on this place.
- Shell script could be created for simplified execution of PHPUnit with a given phpunit.xml configuration.

Coverage reports can be seen in a web browser (Figure 9, Figure 10), a user needs to fill in the URL of the logging path located on the server.
In my case: *http://192.168.179.129/com.scoveco.mm.php/application/tests/coverageReport/index.html*

### 5.2.3 TESTS FOR SCAPPCORE.PHP

Working with transactions here, there is a need to set annotations for test classes to prevent serialization of PDO object that causes crash of tests run [22]:

- @backupGlobals disabled
- @backupStaticAttributes disabled

**Functions under test and the test functions**

The following text lists the public methods (functions) that are present in this class and provides a brief description of their functionality. Each of the function is followed by one or more test functions that verifies the basic functionality.

```
function registerLoginAccount(Account $account)
```

- Check validity of SN account (e.g. FB) by calling SN's API and asking for a details of this account.
- Store contacts from SN.
- Return newly created or already existing identity that has been assigned to this account.

```
test_registerLoginAccount()
```

- Call the function under test using a valid FB account.
- Assert the result - returned value should not be null.

`function loginByAnotherAccount(Account $account)`

- Check validity of SN account using the same process as described in already mentioned function registerLoginAccount .
- Validate if identity status is active.
- Return an identity if already registered (in MM system) and valid account provided.

`test_loginByAnotherAccount_returnsNotNullForRegiteredAcccount()`

- Call the function under test using a valid and already registered FB account. Registration has been done in the previous tests and connection is secured using the annotation "@depends testRegisterLoginAccount" placed before this test's function.
- Assert the result - returned value should not be null.

`test_loginByAnotherAccount_returnsNullForNotYetRegiteredAcccount()`

- Call the function under test using a new instance of Account that has a type set indicating it is a FB account. (Note: If no type set then the NPE appears during tests execution and even PHPUnit cannot return the results of tests run.)
- Assert the result - returned value should be null.

`function createEvent(Event $event)`

- Store an event.
- Return stored event.

`test_createEvent_blankEvent()`

- Call the function under test using a new instance of Event object.
- Assert the result - returned value should not be null.

`function getParticipations(Identity $identity)`

- Return the participations - records from junction table (relationship between identities and events) as an array. This records are created when user creates an event or is invited to any event.

`test_getParticipations_returnsNotNullForRegisteredAccount()`

- Get an identity of valid FB account using function registerLoginAccount and an account from test data source.

- Call the function under test using this identity.
- Assert the result - returned value should not be null.

`test_getParticipations_participationIsCreatedForEventsOwner()`

- Get an identity of registered and valid FB account using the same way as described above.
- Get number of this identity's participations using the method under test.
- Create participation object, assign this identity to it, also set its type to the owner of event.
- Create event object, assign this participation to it, store into DB.
- Assertion - get number of identity's participations, it should be higher than the participations gotten on the beginning of this test.

`test_getParticipations_participationIsCreatedForInvited()`

- The same flow as in the previous test but the participation type is set to invited.

`function updateSNInformations(Identity $identity)`

- Get all registered accounts for given identity and update their details.
- Returns true if everything run well.

`test_updateSNInformations()`

- Get an identity of already registered and valid FB account. This process has been already used in the previous tests.
- Call the function under test using this identity.
- Assert the result - returned value should be true.

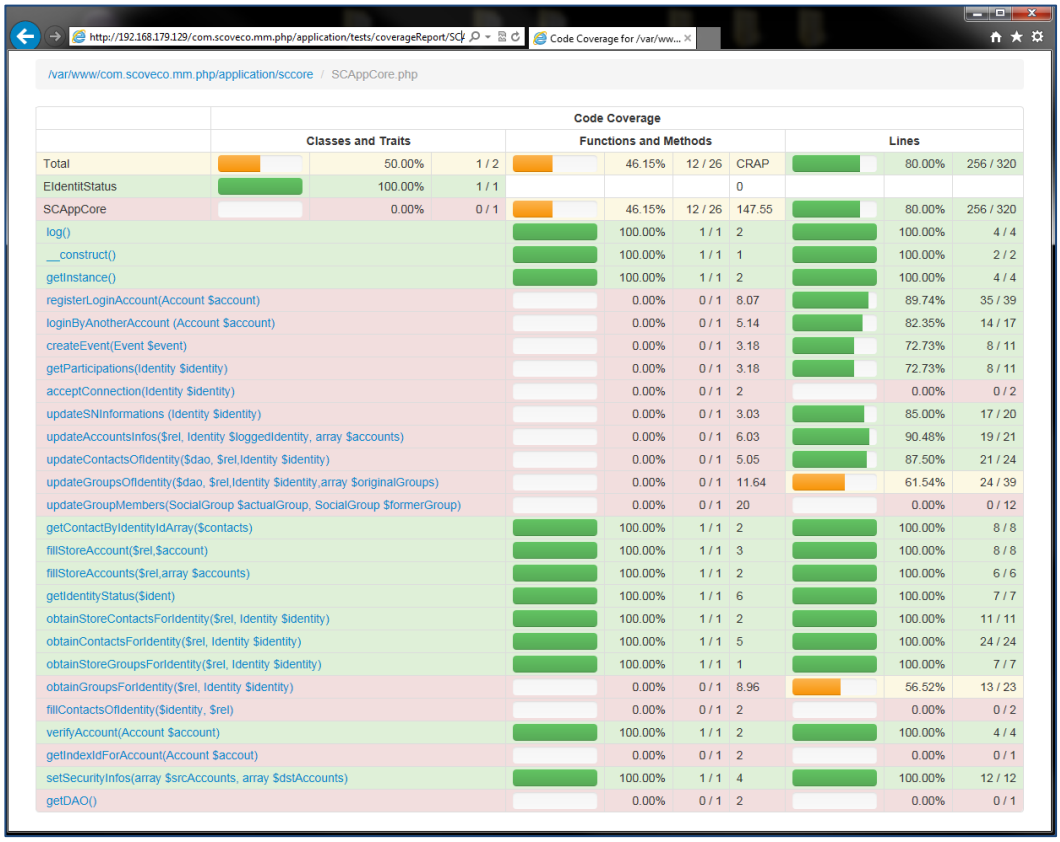



FIGURE 9: SCAPPCORE.PHP CODE COVERAGE

### 5.2.4 TESTS FOR SCDAODOCTRINE.PHP

This class cares about persistence of MM objects (Account, Event...). There are tests for groups of functions created. Again, there is a brief description per each function followed by their tests implementation description.

Note: Variable $rel is the Entity Manager from Doctrine's ORM implementation.

`function crtIdentity($rel)`

- Store a newly generated identity to DB
- Return the stored identity.

`function retIdentities($rel)`

- Return all stored identities.

`function retIdentityByAccountServiceId($rel, Account $account)`

- Return identity for a given account.

`test_crtIdentity_retIdentities()`

- Get number of stored identities using the function under test.
- Create a new identity.
- Commit DB changes.
- Assertion - get current number of stored identities, it should be bigger than on the begging of this test.

`test_retIdentityByAccountServiceId()`

- Create an account containing an identity, set a specific account's first name and service ID.
- Commit DB changes.
- The first assertion - call the function under test using this account, it should return a not null value.
- The second assertion - get accounts from returned identity, one of this accounts should have the same name as the one set on the beginning of this test.

`function crtAccount($rel, Account $account)`

- Store given account to DB.
- Return stored account.

`function updAccount($rel, Account $account)`

- Update a given account to DB.
- Return updated account.

`function retAccountsByIdentityId($rel, $identityId)`

- Return all accounts for a given identity.

`test_retAccountsByIdentityId()`

- Create two accounts for an identity. Remember this identity's ID.
- Assertion - call the function retAccountsByIdentityId using this identity, it should return an array of Accounts with size equals to two.

`test_crtAccount_updAccount()`

- Store a new account to DB.
- Update this account's name.

- Call the function updAccount to update account's data in DB.
- Assert the result - returned value should not be null.

`test_crtAccount_updAccount_retAccountsByIdentityId()`

- Store a new account with identity to DB.
- Update account's first name.
- Assertion - call the function retAccountsByIdentityId, it should contain an account with updated first name.

```
function crtContact($rel,Contact  $contact)
```

- Store a contact into DB.
- Return stored contact.

```
function retContactById($rel,$contactId)
```

- Return contact by ID.

```
function delContact($rel,Contact  $contact)
```

- Delete a given contact from DB.
- Return already deleted account.

`test_crtContact_retContactById()`

- Store a new contact. Function returns stored contact involving a newly generated account ID.
- Assertion - call the function retContactById using acquired account ID, it should not return null.

`test_crtContact_delContact()`

- Store a contact. Remember its ID.
- Delete this contact using function delContact.
- Assertion - call the function retContactById using acquired account ID, it should return null.

```
function crtEvent($rel, Event $event)
```

- Store an event into DB.
- Store participations related to this event.
- Return already stored event.

```
function retEventById($rel, $id)
```

- Return event with given ID.

```
function updEvent($rel, Event $event)
```

- Update event in DB.
- Return updated event.

```
function delEventById($rel, $id)
```

- Delete event from DB.
- Return id of already deleted event.

```
test_crtEvent_updEvent()
```

- Store an event.
- The first assertion - not null value should be returned by function crtEvent
- Set another name to this event object.
- Update this event in DB.
- Assert - call the function retEventById, it should return event with updated name.

```
test_delEventById()
```

- Store an event. Remember its ID.
- Call the function delEventById using this ID.
- Assertion - call the function retEventById to find this event and it should return null value.

```
function crtParticipations($rel,  $parts)
```

- Store participations to DB.
- Return stored participations.

```
function updParticipations($rel,  $parts)
```

- Update participations in DB.
- Return updated participations.

```
function delParticipations($rel,  $parts)
```

- Delete participations from DB.
- Return already deleted participations.

```
test_crtParticipations_updParticipations()
```

- Store participation to DB.

- Update participation object - set its status to any different value.
- Call function updParticipations to store updates to DB.
- Assertion - returned value should contains participation with updated status.

`test_delParticipations()`

- Store participation to DB.
- Call the function delParticipations to delete this participation.
- Assertion - returned value should not be null.

`function crtGroups($rel, $groups)`

- Store groups to DB.

`function updGroups($rel, $groups)`

- Update groups in DB.

`function delGroups($rel, $groups)`

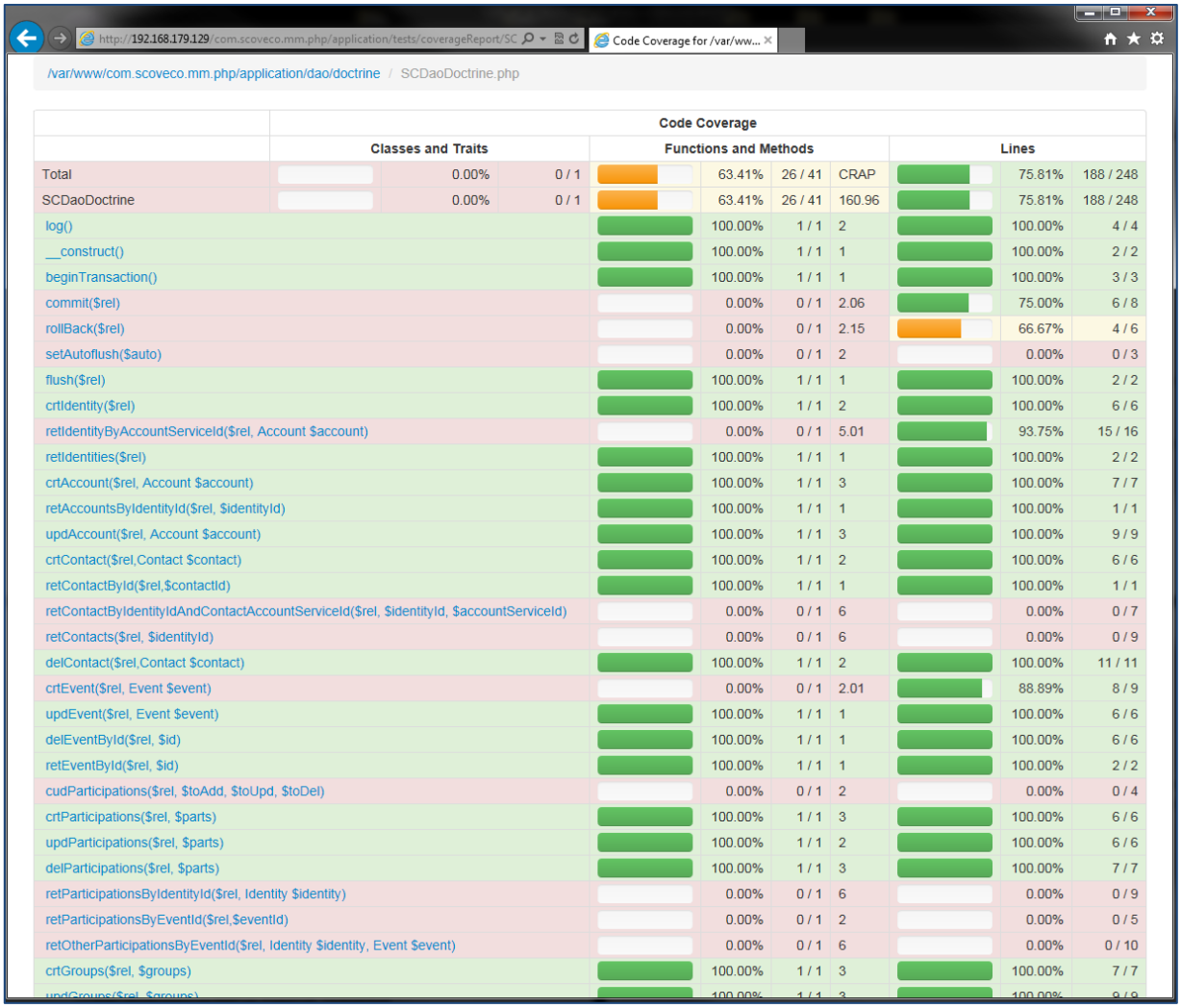
- Delete groups from DB.

`function retGroupsByIdentityId($rel, $identityId)`

- Return array of groups related to given identity.

`test_crtGroups_updGroups_delGroups()`

- Create an account with identity. Remember this identity's ID.
- Create a group and assign this account to it.
- Update this group's name and store it to DB.
- The first assertion - call the function retGroupsByIdentityId using mentioned identity ID, it should return group with updated name.
- Delete this group from DB.
- The second assertion - call the function retGroupsByIdentityId, it should return array of groups with size equals to zero.



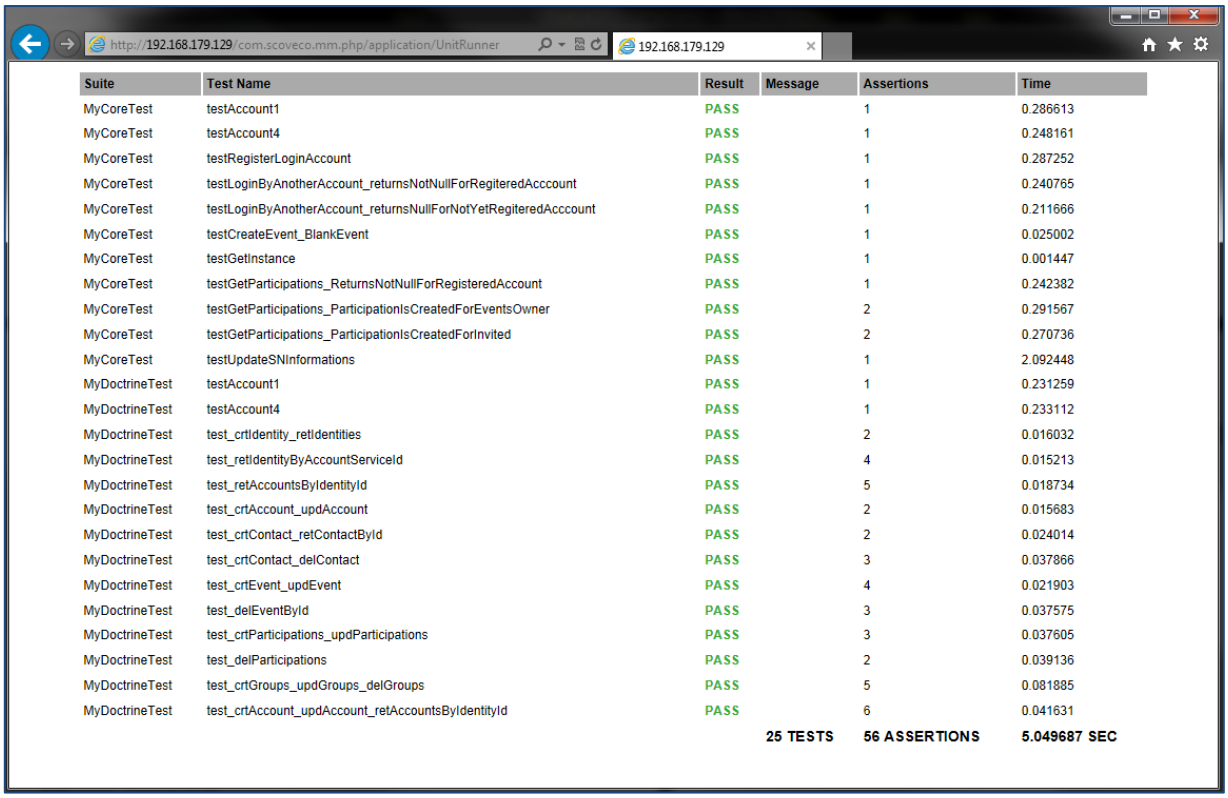

| | Code Coverage | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Classes and Traits | | | Functions and Methods | | | Lines | | |
| Total | | 0.00% | 0 / 1 | | 63.41% | 26 / 41 | CRAP | 75.81% | 188 / 248 |
| SCDaoDoctrine | | 0.00% | 0 / 1 | | 63.41% | 26 / 41 | 160.96 | 75.81% | 188 / 248 |
| log() | | | | | 100.00% | 1 / 1 | 2 | 100.00% | 4 / 4 |
| __construct() | | | | | 100.00% | 1 / 1 | 1 | 100.00% | 2 / 2 |
| beginTransaction() | | | | | 100.00% | 1 / 1 | 1 | 100.00% | 3 / 3 |
| commit($rel) | | | | | 0.00% | 0 / 1 | 2.06 | 75.00% | 6 / 8 |
| rollBack($rel) | | | | | 0.00% | 0 / 1 | 2.15 | 66.67% | 4 / 6 |
| setAutoflush($auto) | | | | | 0.00% | 0 / 1 | 2 | 0.00% | 0 / 3 |
| flush($rel) | | | | | 100.00% | 1 / 1 | 1 | 100.00% | 2 / 2 |
| crtIdentity($rel) | | | | | 100.00% | 1 / 1 | 2 | 100.00% | 6 / 6 |
| retIdentityByAccountServiceId($rel, Account $account) | | | | | 0.00% | 0 / 1 | 5.01 | 93.75% | 15 / 16 |
| retIdentities($rel) | | | | | 100.00% | 1 / 1 | 1 | 100.00% | 2 / 2 |
| crtAccount($rel, Account $account) | | | | | 100.00% | 1 / 1 | 3 | 100.00% | 7 / 7 |
| retAccountsByIdentityId($rel, $identityId) | | | | | 100.00% | 1 / 1 | 1 | 100.00% | 1 / 1 |
| updAccount($rel, Account $account) | | | | | 100.00% | 1 / 1 | 3 | 100.00% | 9 / 9 |
| crtContact($rel,Contact $contact) | | | | | 100.00% | 1 / 1 | 2 | 100.00% | 6 / 6 |
| retContactById($rel,$contactId) | | | | | 100.00% | 1 / 1 | 1 | 100.00% | 1 / 1 |
| retContactByIdentityIdAndContactAccountServiceId($rel, $identityId, $accountServiceId) | | | | | 0.00% | 0 / 1 | 6 | 0.00% | 0 / 7 |
| retContacts($rel, $identityId) | | | | | 0.00% | 0 / 1 | 6 | 0.00% | 0 / 9 |
| delContact($rel,Contact $contact) | | | | | 100.00% | 1 / 1 | 2 | 100.00% | 11 / 11 |
| crtEvent($rel, Event $event) | | | | | 0.00% | 0 / 1 | 2.01 | 88.89% | 8 / 9 |
| updEvent($rel, Event $event) | | | | | 100.00% | 1 / 1 | 1 | 100.00% | 6 / 6 |
| delEventById($rel, $id) | | | | | 100.00% | 1 / 1 | 1 | 100.00% | 6 / 6 |
| retEventById($rel, $id) | | | | | 100.00% | 1 / 1 | 1 | 100.00% | 2 / 2 |
| cudParticipations($rel, $toAdd, $toUpd, $toDel) | | | | | 0.00% | 0 / 1 | 2 | 0.00% | 0 / 4 |
| crtParticipations($rel, $parts) | | | | | 100.00% | 1 / 1 | 3 | 100.00% | 6 / 6 |
| updParticipations($rel, $parts) | | | | | 100.00% | 1 / 1 | 2 | 100.00% | 6 / 6 |
| delParticipations($rel, $parts) | | | | | 100.00% | 1 / 1 | 3 | 100.00% | 7 / 7 |
| retParticipationsByIdentityId($rel, Identity $identity) | | | | | 0.00% | 0 / 1 | 6 | 0.00% | 0 / 9 |
| retParticipationsByEventId($rel,$eventId) | | | | | 0.00% | 0 / 1 | 2 | 0.00% | 0 / 5 |
| retOtherParticipationsByEventId($rel, Identity $identity, Event $event) | | | | | 0.00% | 0 / 1 | 6 | 0.00% | 0 / 10 |
| crtGroups($rel, $groups) | | | | | 100.00% | 1 / 1 | 3 | 100.00% | 7 / 7 |
| updGroups($rel, $groups) | | | | | 100.00% | 1 / 1 | 3 | 100.00% | 9 / 9 |

FIGURE 10: SCDAODOCTRINE.PHP CODE COVERAGE

## 5.3 TEST RUNNER RESULTS REPRESENTATION

The test runner script uses the API of *PHPUnit_Framework_TestSuite* to group the tests into test suites, executes them and collects the test run results. The other part of this script cares about rendering the results into a table and saves it into html files [24].

Results of my tests - *http://192.168.179.129/com.scoveco.mm.php/application/UnitRunner* are shown on the Figure 11.




| Suite | Test Name | Result | Message | Assertions | Time |
|---|---|---|---|---|---|
| MyCoreTest | testAccount1 | PASS | | 1 | 0.286613 |
| MyCoreTest | testAccount4 | PASS | | 1 | 0.248161 |
| MyCoreTest | testRegisterLoginAccount | PASS | | 1 | 0.287252 |
| MyCoreTest | testLoginByAnotherAccount_returnsNotNullForRegiteredAcccount | PASS | | 1 | 0.240765 |
| MyCoreTest | testLoginByAnotherAccount_returnsNullForNotYetRegiteredAcccount | PASS | | 1 | 0.211666 |
| MyCoreTest | testCreateEvent_BlankEvent | PASS | | 1 | 0.025002 |
| MyCoreTest | testGetInstance | PASS | | 1 | 0.001447 |
| MyCoreTest | testGetParticipations_ReturnsNotNullForRegisteredAccount | PASS | | 1 | 0.242382 |
| MyCoreTest | testGetParticipations_ParticipationIsCreatedForEventsOwner | PASS | | 2 | 0.291567 |
| MyCoreTest | testGetParticipations_ParticipationIsCreatedForInvited | PASS | | 2 | 0.270736 |
| MyCoreTest | testUpdateSNInformations | PASS | | 1 | 2.092448 |
| MyDoctrineTest | testAccount1 | PASS | | 1 | 0.231259 |
| MyDoctrineTest | testAccount4 | PASS | | 1 | 0.233112 |
| MyDoctrineTest | test_crtIdentity_retIdentities | PASS | | 2 | 0.016032 |
| MyDoctrineTest | test_retIdentityByAccountServiceId | PASS | | 4 | 0.015213 |
| MyDoctrineTest | test_retAccountsByIdentityId | PASS | | 5 | 0.018734 |
| MyDoctrineTest | test_crtAccount_updAccount | PASS | | 2 | 0.015683 |
| MyDoctrineTest | test_crtContact_retContactById | PASS | | 2 | 0.024014 |
| MyDoctrineTest | test_crtContact_delContact | PASS | | 3 | 0.037866 |
| MyDoctrineTest | test_crtEvent_updEvent | PASS | | 4 | 0.021903 |
| MyDoctrineTest | test_delEventById | PASS | | 3 | 0.037575 |
| MyDoctrineTest | test_crtParticipations_updParticipations | PASS | | 3 | 0.037605 |
| MyDoctrineTest | test_delParticipations | PASS | | 2 | 0.039136 |
| MyDoctrineTest | test_crtGroups_updGroups_delGroups | PASS | | 5 | 0.081885 |
| MyDoctrineTest | test_crtAccount_updAccount_retAccountsByIdentityId | PASS | | 6 | 0.041631 |
| | | | 25 TESTS | 56 ASSERTIONS | 5.049687 SEC |

FIGURE 11: UNITRUNNER'S RESULTS IN A WEB BROWSER

# 6 TESTS IMPLEMENTATION FOR THE CLIENT

The task here is to automate the TCs for manual system testing specified in Section 4.2. The objective is not to strictly follow the flow of manual tests, the steps can be adapted for automation or the assertions can be reduced or even expanded to profit from information processing by a PC (and not by a human).

## 6.1 SETUP TESTING ENVIRONMENT

System tests are being solved here therefore a whole MM system needs to be setup for the test. The server part has been already configured in Section 5.1.1. Client is the only missing part now. In a real use there will be an Android mobile device with MM client installed. For development purpose is better to use an emulator which is part of Android SDK.

### 6.1.1 SETUP WORKING ENVIRONMENT FOR ANDROID

- Install JDK.
- Download and unpack Android SDK.
- Install IDE → Eclipse.
- Install ADT plug-in into Eclipse.
- Let the Eclipse know where the Android SDK is located using Eclipse's Preferences.

Using the ADT is a user able to control the Android SDK and Android emulators (AVD) using the Eclipse IDE interface. After ADT has been installed and Eclipse restarted, there are two new icons in Eclipse menu shown (Figure 12), i.e. *Android SDK manager* and *Android AVD manage.*

- Install the latest packages for Android SDK using the Android SDK manager.
- Setup an AVD using Android AVD manager. This emulator is used when the user starts execution of automated tests.

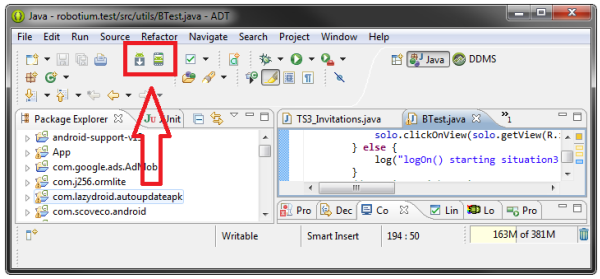



FIGURE 12: ADT INSTALLED INTO ECLIPSE

**Emulator performance**

Android emulator must emulate ARM instructions when running on the x86 based host. This operation is highly processor intensive, emulator seems to be slow and its performance could be a bottleneck when developing the tests.

The first idea I've found was to set the affinity of emulator process to a CPU core different to CPU0 what makes sense since the CPU0 cares for the other processes running on the workstation but this didn't help much.

The HAXM from Intel is a good way how to speed up the emulator but currently it is available for Intel CPUs only and my workstation contains AMD CPU. Also there is still no alternative from the AMD side [25].

Androidx86 was the solution for me. More about in the Section 6.1.2

### 6.1.2 ANDROIDX86 IN VIRTUALBOX

Android-x86 is a port of the Android mobile operating system to the x86 instruction set architectures. This projects began as series of patches to the Android source code to enable it to run on various devices using x86 processors [26].

Comparing to the AVD the performance and responses are much better. This is the reason why to use it while testing scripts are being developed and debugged.

Configuration:

- Install a virtualization product → VirtualBox.
- Download the ISO image file for Androidx86 (the current version 4.2).
- Install the Androidx86 system into the VirtualBox.
- Setup ports forwarding for Eclipse debugger (Figure 13).
- Run the Androidx86.
- Map running Androidx86 to ADB, a batch file can be useful for repeated usage:
    - *adb connect 158.196.42.130:5555*

After this configuration done, the Eclipse knows about Androidx86 machine. If tests execution is triggered the Eclipse deploys an apk file for the application and the another one for testing project into the Androidx86 machine. It also starts performing test after deployment has been done [26].

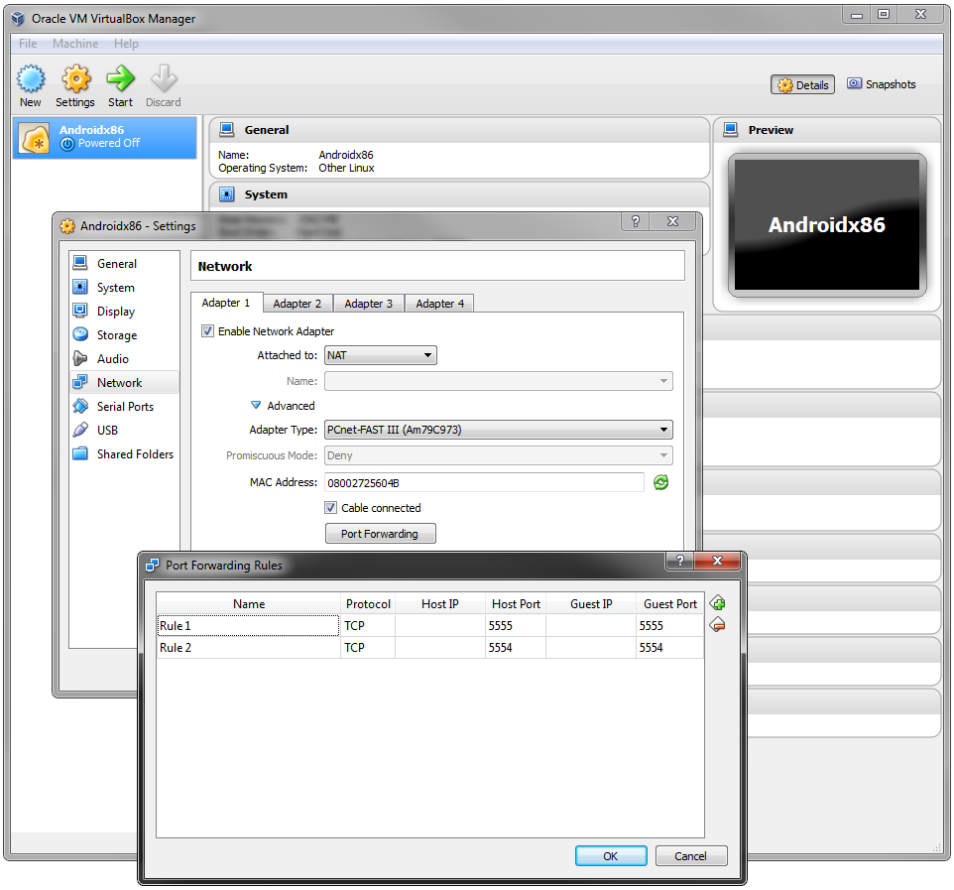



FIGURE 13: PORT FORWARDING FOR ANDROIDX86

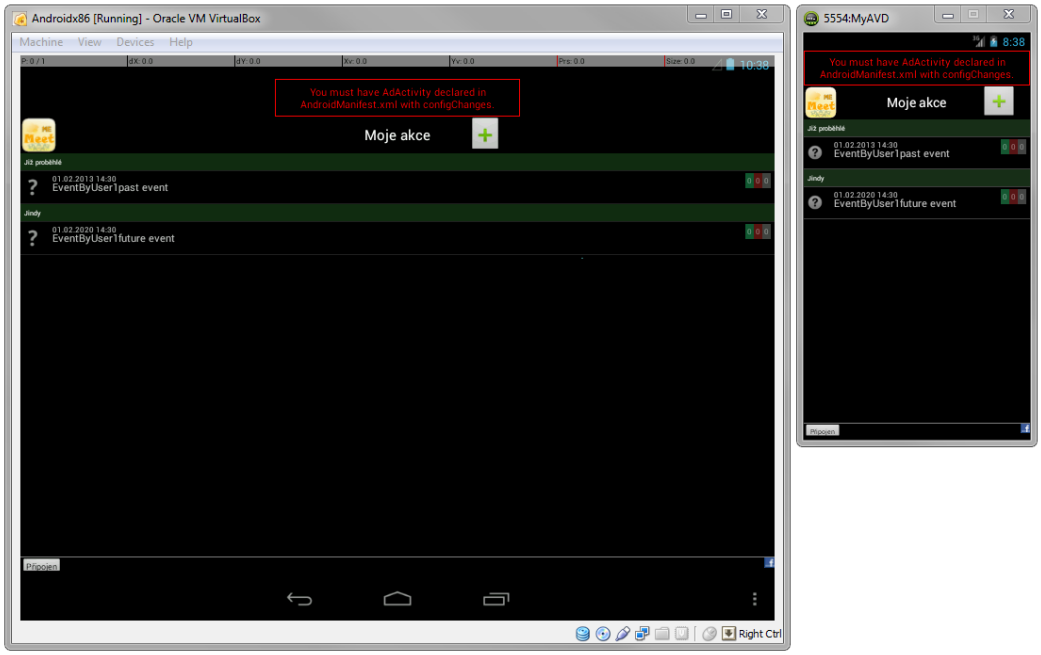



FIGURE 14: ANDROISX86V4.2 IN VIRTUALBOX VS. ANDROIDV4.2 IN AVD

## 6.2 IMPLEMENTATION

*"Robotium is an Android test automation framework that has full support for native and hybrid applications. Robotium makes it easy to write powerful and robust automatic black-box test cases. With the support of Robotium, test case developers can write function, system and acceptance test scenarios, spanning multiple Android activities."* [19]

### 6.2.1 TEST PROJECT SETUP AND THE TECHNIQUES IN USE

New Eclipse project will be created for automated tests. Test will be represented by a public methods organized in JUnit classes.

**Basic configuration**

- Create a new Android Test Project in Eclipse.
- Select test target → MM client project imported in workspace.
- Select minimal SDK version (Android 2.2).
- Add Robotium jar into project's build path.
- Create a new JUnit 3 TC class.
- Create test methods in already created TC class.

**Binding the client with a server**

Before the tests execution, the client app needs to be configured. Add a new server (the current testing server set in Section 5.1.1) in the MM client's *Configuration.java* and set it as active server in *IMMServerConfigs.java.*

**Disable Key Guard**

AVD starts with a locked screen as default. The AUT cannot receive key events and ATC fails. It needs to be unlocked manually but there is also a programmatic way:

- Add *DISABLE_KEYGUARD* permission into manifest file for the MM client.
- Disable Key Guard Lock in the setUp method.

AVD still starts with a locked screen but whenever a setUp method is invoked before the test, it unlocks the screen and the test can run without this problem [28].

### ATC prototype

It contains parts that repeats in majority of particular TCs. All the TCs extends this prototype.

Recurring parts:

- Fixture methods → setUp, tearDown.
- Android TC constructor.
- Constants holding test accounts.
- Other custom methods (for messages log, etc.)

### Fixture setup

Expected precondition for each test is to have no MM Event on the startup. This is solved by the setUp method that uses MM functionality to delete an Event in a loop while any Event present in the list of user's Events (My Events list).

### Concept of Conditions

User defines a condition and a state when this condition is satisfied. Later, in code for tests, this is used as Boolean value with a specific timeout interval. The runtime waits for the specific time for condition to be satisfied and returns true, if not satisfied and the timeout is exceeded then it returns false [27]. It solves the timing issue.

### GUI items locators

The basic Robotium approach is to use BlackBox techniques. App's objects are identified according to the labels (e.g. a button with label "OK") or indexes (e.g. there are two button on the screen, the first button is identified by "button[0]"). User can create this locaters without a knowledge of AUT's source code. This method simplifies the test development from the beginning but makes the code less readable, also mistakes can appear when the app's view is bigger than the screen and scrolling is needed.

More reliable approach is to use the resource IDs that are generated by Android SDK in AUT's project. However it violates the BlackBox approach but it seems to be more useful.

### Workaround to recognize My Events and Friends Events views

From the MM client's architecture, the Home view (the first view or screen shown after login) is represented by My Events view. Another view - Friends Events view appears when user scrolls to the right. Problem is to recognize which view is currently shown.

A solution done using Conditions. If user needs to work with My Events view then a process to scroll to the left is invoked (even if this view is already shown) in Conditions and this Condition immediately returns true after the scroll done. Similarly, there is another Condition created for Friends Events view.

**WebView**

Login architecture for MM system is described in Section 3.3.2. This architecture causes the MM client is a hybrid app since it contains a WebView (Figure 15) to present the FB login form. The Robotium is able to handle it just from its 4.0 release version that was released during my work on this tests [29]. This was a big problem for automation since Robotium was not able to pass through this and could not access the Home screen of AUT (naturally, this is a starting point for testing and automation also).

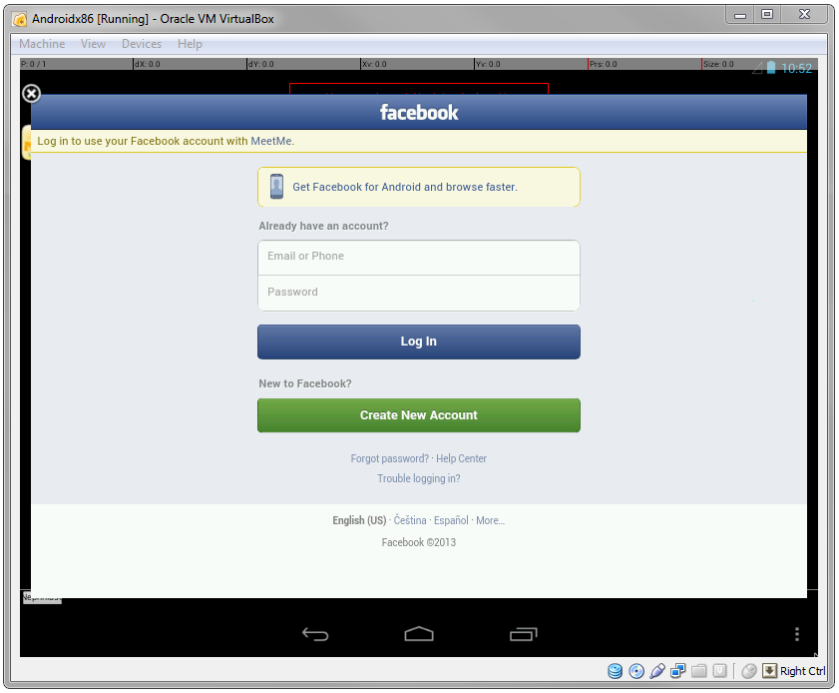



FIGURE 15: WEBVIEW INJECTED INTO MM APP

The previous solution was to set the MM client to a mode omitting the FB login form. Account value was hardcoded into MM client's code and it was not possible to automate scenarios describing interactions between MM Users (the MM Events Invitations).

Robotium 4.0 is able to handle the FB login form, therefore the test flows containing switching between users can be automated.

**Example of the basic ATC**

The following JUnit class shows a basic ATC to test the login functionality. Main parts are:

- Solo initialization.
- SetUp, TearDown methods definition.
- The test with assertions (there can be more test methods within a test class).

```
public class FirstTest extends
ActivityInstrumentationTestCase2<MeetMeActivity> {

    // Solo provides an API providing all GUI user actions
    private Solo solo;

    public FirstTest() {
    // Connection to the AUT
    super(MeetMeActivity.class);
    }
    // This executes before every test method
    protected void setUp() throws Exception {
    super.setUp();
    solo = new Solo(getInstrumentation(), getActivity());
    }
    // This executes after every test method
    protected void tearDown() throws Exception {
    solo.finishOpenedActivities();
    super.tearDown();
    }

    // Specific test
    public void testSimpleLogin() {
    // Continue with FB icon
    solo.clickOnView(solo.getView(R.id.imageButton1));

    // Fill in the user's email
    solo.waitForWebElement(By.name("email"));
    solo.clickOnWebElement(By.name("email"));
    solo.enterTextInWebElement(By.name("email"), "hol1@centrum.sk");

    // Fill in the user's password
    solo.waitForWebElement(By.name("pass"));
    solo.clickOnWebElement(By.name("pass"));
    solo.enterTextInWebElement(By.name("pass"), "butter");

    // Click the login button
    solo.waitForWebElement(By.name("login"));
    solo.clickOnWebElement(By.name("login"));
    solo.sleep(5000); // TODO: Needs to be replaced with dynamic wait.

    // Assert the user has been successfully connected
    Assert.assertTrue("Status bar does not show *Connected* after login.",
       solo.searchText(solo.getString(R.string.statusBar_connected_text)));
    }

}
```

Note: The previous example is a simplified version of the implemented ATC that is in use for regression tests since there are many additional checks needed for dialog windows that could appear, etc.

### 6.2.2 REUSABLE LOGIN FLOW

This flow is used per each ATC therefore it is implemented in the TC prototype. ATCs extends this prototype and can access this flow (method). Basically it is the extended version of the previously shown example of ATC, but there are more check for various statuses of MM app (already saved session, etc.).

### 6.2.3 TEST SUITE TO CHECK THE TESTING ACCOUNTS

The testing accounts are used in all ATCs. Their validity must be checked at the start of each regression tests run since all ATCs will fail if any of this accounts is invalid.

This ATC is a JUnit class to check the test data - FB accounts described in Section 3.3.3. There are two test methods for two accounts. Assertions is satisfied when a status button of MM app has a label "Connected". It means the ATC passed through FB login and the MM app is able to accept this account. There are two ATCs implemented: ATC0.1, ATC0.2.

### 6.2.4 TEST SUITE TO TEST MY EVENTS - CRUD

There are four test methods to check the basic operations with MM Events. The following text describes flows used for implementation of this ATCs.

**ATC2.1: Create Event (covers TC2.1)**

- Actor logs in.
- Actor clicks the Create Event button.
- Actor fills in the Events name input field.
- Actor click the Save Event button.
- Assertion → Event's name should be shown in the list of My Events.

**ATC2.2: Update Event (covers TC2.2)**

- Reused flow for Event creation → ATC2.1.
- Actor clicks the already created Event's item → Event's details view appears.
- Actor updates the Event's name.
- Actor clicks the Save Event button.
- Assertion → Updated Event's name should be shown in the list of My Events.

**ATC2.3: Delete Event (covers TC2.3)**

- Reused flow for Event creation → ATC2.1.
- Actor clicks and holds on already created Event's item →Dialog to Delete this item appears.
- Actor confirms the dialog.
- Assertion → Deleted Event's name should not be shown in the list of My Events.

**ATC2.4: Create a fully specified Event (extended test flow)**

- Actor logs in.
- Actor clicks the Create Event button.
- Actor specifies all the Event's details.
- Actor click the Save Event button.
- Actor logs out.
- Actor logs in.
- Actor opens the Event.
- Assertion per each Event's detail.

### 6.2.5 TEST SUITE TO TEST INVITATIONS

**ATC3.0: View invitation (reusable flow for ATC3.1, ATC3.2)**

- Actor1 logs in.
- Actor1 creates an Event and invites the Actor2 → similar to ATC2.1
- Actor1 logs out.
- Actor2 logs in.
- Assertion → Invitation present in the list of Friends Events.

**ATC3.1: Accepts invitation (covers TC3.1)**

- Reused flow ATC3.0.
- Actor2 accepts the invitation.
- Assertion1 → Invitation still present in the list of Friends Events.
- Actor2 logs out.
- Actor1 logs in.
- Assertion → Number of accepted.

**ATC3.2: Decline invitation (covers TC3.2)**

- Reused flow ATC3.0.
- Actor2 declines the invitation.

- Assertion1 → Invitation no more present in the list of Friends Events.
- Actor2 logs out.
- Actor1 logs in.
- Assertion2 → Number of declined.

**ATC3.3: Past invitation (extended test flow)**

Used flows similar to already mentioned. But the Actor1 sets the date for the Invitation to the past. Assertion - Actor2 cannot see this Event in the list of Friends Events.

**ATC3.4: Deleted invitation (extended test flow)**

Actor1 creates an Event and subsequently deletes it. Actor2 should not see it in the list of Friends Events.

### 6.2.6 ADDITIONAL ATCS

There are more ATCs that extends the tests. The same as already mentioned methods are used, just the combinations and ordering differ. Their purpose is straightforward from their names:

- ATC4.1: Event's types fetching
- ATC4.2: Friends fetching
- ATC5.1: Events grouping

## 6.3 ATCS EXECUTION

There is a JUnit class representing the set of tests for regression testing. ATCs can be added using the TestSuite API that comes from JUnit framework.

The current process of executing this regression tests is manual:

- User starts Eclipse. The testing project and the project for the MM client must be correctly imported.
- User maps an MM server to the MM client.
- User clears the mapped DB (at least deletes all MM Events related to the test accounts).
- User starts an Android emulator and maps it to the ADB (already mentioned in Section 6.1.2)
- User selects wanted test or the class which represents the set of regression tests and runs it as Android JUnit Test.

Note: Regression tests should be run on the AVD which is closer to the real devices when comparing to the Androidx86. Androidx86 is still buggy and tests runs are very unstable.

# 7 CONCLUSION

**Testing process includes configuration of testing environment**

The first task of this work was to get running system for which the tests were implemented subsequently. This can be simple for an independent desktop application but the client - server architecture with a native, mobile client makes things a bit complicated. In this case the configuration, involving possibility to debug, can take actually days for a novice.

**Integration testing is a good practice**

I run through several web forums and literature related to Unit and Integration testing during the work on this thesis and Unit testing is very recommended. However, the practice shows that it is usually neglected or forgotten. The proper Unit tests are currently missing for this system too but it would take a lot of effort to implement it now. My work here was to create Integration tests for selected classes and it seems to be a good patch for missing Unit tests. With a lesser effort the tests are created for a class integrating some functionality only, but in fact, these tests check classes related (integrated) also. Needles to say, the previous statement does not signify the needles of integration tests if unit tests already exist.

From metrics used to get measurable level of test automation was used the line coverage. The current tests implementation covers 80% lines of code of tested classes. There are more reliable metrics (e.g. branch coverage) but the line coverage is the only one provided by PHPUnit tool.

Drawback here is the bigger amount of entrance knowledge needed to be able to create this tests. An author needs to be/get familiar with PHP, internal logic of application and the code. It seems to be best to create this tests by a developer of the application under tests.

**System tests automation is not always effective**

From my practice as a QA Engineer I'm experienced with System tests automation for web applications and it usually runs very smoothly. The client here is a native, mobile application therefore the mature tools for web applications cannot be used. Actually there are also tools for this sort of applications but identification of GUI objects is harder. Emulators seems to be slow and sometimes unexpected situation can appear. ATCs creation often reshapes to finding workarounds.

Run of the current set of ATCs takes over fifty minutes. This runs are unstable and results are not reliable. Manual tests shouldn't take more than ten minutes. It seems this application is not good

candidate for automation of System tests. However the ATCs could be improved, alternatively trimmed to be more reliable. Robotium and emulators have also place to be better.

This practice still can be useful. Applications with simple GUI layouts, limited amount of navigation between screens are good candidates.

**Future work**

Improving stability of automated System tests - the tests for the MM client on the usable level. Either the flows could be simplified or the algorithms improved.

Fixtures setup could be much better. Currently all the implemented ATCs expects truncated DB tables at the startup. A simple and rapid mechanism for DB purging should be introduced into setUp methods. This would improve the ATCs especially for the MM client since the current setUp method logs in both of test users and tries to delete all their Events. This prolongs the tests runs by minutes and increases a risk of unexpected failure.

Nowadays all the ATCs needs to be started manually but the execution can be also automated and scheduled. Continuous Integration tools are monitoring and managing jobs and planned regression tests execution can be also set as a job for a CI tool (e.g. Jenkins).

# REFERENCE LIST


1. ASTQB. *ISTQB Foundation Level Certification Syllabus*, 2011. Available from: <http://www.astqb.org/get-certified/istqb-syllabi-the-istqb-software-tester-certification-body-of-knowledge/>. [21 April 2013].
2. GALIN, D. *Software Quality Assurance: From Theory to Implementation*. 1 ed.: Addison-Wesley, 2003. 616p. ISBN 0-201-70945-7.
3. WILLIAMS, L. *A (Partial) Introduction to Software Engineering Practices and Methods,* NCSU, 2008-2009. [online]. Available from: < http://www.cs.umd.edu/~mvz/cmsc435-s09/pdf/Williams-draft-book.pdf>. [21 April 2013].
4. *Android : Connecting to MySQL using PHP*. [online]. Available from: <http://madhusudhanakn.wordpress.com/2011/05/19/ANDROID-CONNECTING-TO-MYSQL-USING-PHP/>. [21 April 2013].
5. *PHP Manual*. [online]. Available from: <http://cz2.php.net/manual/en/index.php>. [21 April 2013].
6. *XML-RPC Specification*. [online]. Available from: <http://xmlrpc.scripting.com/spec.html>. [21 April 2013].
7. *Ubuntu Server Guide*. [online]. Available from: <https://help.ubuntu.com/12.10/serverguide/serverguide.pdf>. [21 April 2013].
8. *Application Fundamentals*. [online]. Available from: <http://developer.android.com/guide/components/fundamentals.html>. [21 April 2013].
9. MCCRACKEN, H. *Who's Winning, iOS or Android? All the Numbers, All in One Place*. [online]. Available from: <http://techland.time.com/2013/04/16/ios-vs-android/>. [21 April 2013].
10. *Android open source project*. [online]. Available from: <http://source.android.com/index.html>. [21 April 2013].
11. SMITH, A. *Facebook reaches one billion users*. [online]. Available from: <http://money.cnn.com/2012/10/04/technology/facebook-billion-users/index.html>. [21 April 2013].
12. *Facebook SDK for Android*. [online]. Available from: <https://developers.facebook.com/android/>. [21 April 2013].

13. *Facebook Login*. [online]. Available from: < https://developers.facebook.com/docs/facebook-login/>. [2 May 2013].
14. *Test plan overview.* [online]. Available from: <http://publib.boulder.ibm.com/infocenter/rqmhelp/v1r0m0/index.jsp?topic=%2Fcom.ibm.rational.test.qm.doc%2Ftopics%2Fc_planning_overview.html>. [21 April 2013].
15. *Getting Started - About Version Control.* [online]. Available from: <http://git-scm.com/book/en/Getting-Started-About-Version-Control>. [21 April 2013].
16. HAYES, G. L. *The Automated Testing Handbook.* [online]. Available from: <http://www.softwaretestpro.com/ItemAssets/4772/AutomatedTestingHandbook.pdf>. [21 April 2013].
17. FOWLER, M. *XUnit.* [online]. Available from: < http://www.martinfowler.com/bliki/Xunit.html>. [21 April 2013].
18. *Robotium Tutorials.* [online]. Available from: <https://code.google.com/p/robotium/wiki/RobotiumTutorials>. [21 April 2013].
19. *Robotium Project.* [online]. Available from: <https://code.google.com/p/robotium/>. [21 April 2013].
20. *Documentation - Stuff to Read.* [online]. Available from: <http://curl.haxx.se/docs/>. [21 April 2013].
21. *Setting Up Remote Debugging*. [online]. Available from: <http://files.zend.com/help/PDT/troubleshooting_remote_debugging.htm>. [21 April 2013].
22. *PHPUnit Manual.* [online]. Available from: <http://phpunit.de/manual/3.8/en/automating-tests.html>. [21 April 2013].
23. *Documentation for: XDebug 2.* [online]. Available from: <http://xdebug.org/docs/>. [21 April 2013].
24. *PHPUnit_Framework_TestSuite.* [online]. Available from: <http://pear.php.net/reference/PHPUnit2-2.0.0beta1/phpunit.framework/PHPUnit_Framework_TestSuite.html>. [21 April 2013].
25. DARCEY, L. *Supercharge Your Slow Android Emulator*. [online]. Available from: <http://www.developer.com/ws/android/development-tools/supercharge-your-android-emulator-speed-with-intel-emulation-technologies.html>. [4 May 2013].
26. *Android-x86 Project - Run Android on Your PC*. [online]. Available from: < http://www.android-x86.org/>. [21 April 2013].

27. *Robotium - Waiting for an Activity's tasks to complete*. [online]. Available from: <http://stackoverflow.com/questions/14689422/robotium-waiting-for-an-activitys-tasks-to-complete>. [21 April 2013].
28. *Activity Testing Tutorial*. [online]. Available from: <http://developer.android.com/tools/testing/activity_test.html>. [21 April 2013].
29. RENAS, R. *Robotium 4.0 – Web Support!* [online]. Available from: <http://www.jayway.com/2013/03/04/robotium-4-0-web-support/>. [21 April 2013].
30. PATTON, R. *Software Testing*. Indiana: Sams Publishing, 2001. 389p. ISBN 0-672-319837.
31. OSHEROVE, R. *The art of Unit testing*. Manning Publications Co., 2011. 324p. ISBN 978-1-933988-27-6.

# LIST OF ATTACHMENTS

A. Source codes for ATCs placed on the attached CD.